算法的时间复杂度

当算法的规模 n 增大时，时间的变化情况，记为 $T(n)=O(f(n))$，算法的时间增长与语句的频度有关。下列 f 为语句频度的表达函数

$O(1)\ f$: printf(""); $\lim\limits_{n\to n_0} f(n)\to 1$

$O(n)\ f$: for(int i=0; i<n; i++) printf(""); $\lim\limits_{n\to n_0} f(n)\to n_0$

我们分析一些特别的例子 $\sum\limits_{i=0}^{n} 1 = n$

1.
```
for(int i=0;i<n;i++)
    for(int j=0;j<i;j++)
        printf("");
```
易知 $\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{i} 1 = \sum\limits_{i=0}^{n} i = \frac{(0+n)n}{2}$

故有 $\lim\limits_{n\to n_0} f(n)\to \frac{1}{2}n^2 = O(n^2)$

2.
```
int i=1;
while(i<n) i*=2;
```
分析次数设为 x，$2^x \geq n$ 则 $x \geq \log_2 n$ $(n>0)$

同理分析得 $x \leq \log_2 n-1$，$f(n)=x$

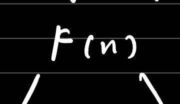
故 $\log_2 n-1 \leq f(n) \leq \log_2 n$ 跟据夹逼准则

当 $\lim\limits_{n\to n_0} f(n) \to \log_2 n_0 = O(\log_2 n)$

3. Fib 数列

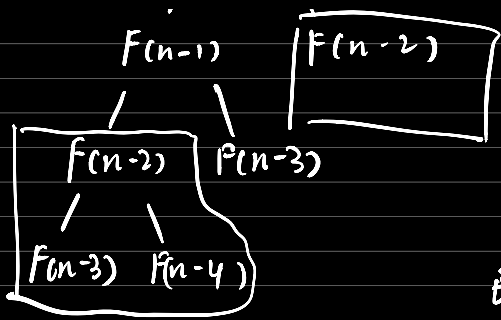
有 $\begin{cases} f(n)=f(n-1)+f(n-2) \\ f(0)=f(1)=1 \end{cases}$

1. 二叉树分析（调用栈）



假设 $n=4$，则树的高度为4

我们 T(n) 在 n 大相当一部分子树



我们可以看出 有相当一部分的节点是重复计算的，因此不仅看出 $f(n) < O(\log_2 n)$，就此种规律而言我们并不能用级数描述其规律 即 $\sum_{i=0}^{n} Fib(i)$ 表达起来会很困难。因此直接视其复杂度为 n 和 $\log_2 n$ 之间

2. $\begin{cases} f(n) = f(\frac{n}{2}) + 1 \\ f(1) = 1 \end{cases}$ 反复代入

$$
\begin{aligned}
f(n) &= f(\tfrac{n}{2}) + 1 \\
&= f(\tfrac{n}{4}) + 2 \\
&= f(\tfrac{n}{2^i}) + i \\
&= \log_2 n + f(1)
\end{aligned}
$$

即其复杂度为 $\log_2 n$，从递归调用分析上也能轻易看出其为一颗线性的树.

空间复杂度：

相比于时间复杂度而言，空间复杂度要相对简单，其为规模为 n，所占有的内存为多少。其分析难度比之时间要简单很多. 一般对于调用栈开辟新空间可用尾递归解决问题。